Performance and Fault Tolerance of Preconditioned Iterative Solvers on Low-Power ARM Architectures

Performance and Fault Tolerance of Preconditioned Iterative Solvers on Low-Power ARM Architectures

José I. ALIAGA , [a,1] Sandra CATALÁN, [a] Charalampos CHALIOS, [b]
Dimitrios S. NIKOLOPOULOS, [b] Enrique S. QUINTANA-ORTÍ [a]
[a] *Dpto. Ingeniería Ciencia de Computadores, Universidad Jaume I, Castellón (Spain)*
[b] *School of EEECS, Queen's University of Belfast (United Kingdom)*

Abstract. As the complexity of computing systems grows, reliability and energy are two crucial challenges that will demand holistic solutions. In this paper, we investigate the interplay among concurrency, power dissipation, energy consumption and voltage-frequency scaling for a key numerical kernel for the solution of sparse linear systems. Concretely, we leverage a task-parallel implementation of the Conjugate Gradient method, equipped with an state-of-the-art preconditioner embedded in the ILUPACK software, and target a low-power multicore processor from ARM. In addition, we perform a theoretical analysis on the impact of a technique like Near Threshold Voltage Computing (NTVC) from the points of view of increased hardware concurrency and error rate.

Keywords. Sparse linear systems, iterative solvers, ILUPACK, low power multicore processors, high performance, energy efficiency, convergence

1. Introduction

As we move along the scaling projection for computing systems predicted by Moore's law, some of the technologies that have fuelled this exponential growth seem to be heading for serious walls enforced by physical constraints [7]. Concretely, a system with billions of components will experience multiple faults and, therefore, the software has to be made resilient in order to deal with this scenario. In addition, with the end of Dennard's scaling [5], the fraction of silicon that can be active (at the nominal operating voltage) for a target thermal design power (TDP) rapidly decays. As a consequence, computer architectures have turned towards dark silicon and heterogeneous designs [8], and Near Threshold Voltage Computing (NTVC) has arisen as an appealing technology to reduce energy consumption at the cost of increasing error rates.

The *High Performance Conjugate Gradients (HPCG) benchmark* [6] has been recently introduced in an effort to create a relevant metric for ranking HPC systems using a benchmark with data access patterns that mimic those present in crucial HPC applications. In the reference implementation of HPCG, parallelism is extracted via MPI

[1]Corresponding Author: Dpto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I, 12.071–Castellón (Spain); E-mail: aliaga@icc.uji.es.

and OpenMP [6]. However, in an era where general-purpose processors (CPUs) contain dozens of cores, the concurrency that is targeted by this legacy implementation may be too fine-grain.

In this paper we investigate the scalability, energy efficiency and fault resilience of low-power multicore ARM processors using our task-parallel version [1] of ILUPACK[2] (Incomplete LU PACKage). This is a multi-threaded CG solver for sparse linear systems furnished with a sophisticated algebraic multilevel factorization preconditioner. Compared with the HPCG benchmark, our implementation of ILUPACK exhibits analogous data access patterns and arithmetic-to-memory operation ratios. On the other hand, our version is likely better suited to exploit the hardware concurrency of current multicore processors [1,2]. This paper is an extension of previous work [4], with the following major differences:

- First, we target a much more sophisticated iterative solver, with a complex preconditioner based on ILUPACK, instead of the simple CG iteration in [4].
- Second, we employ a task-parallel version of the solver, in lieu of the simple loop-based parallelization of the numerical kernels integrated into CG that was leveraged in [4].
- As a result, our task-parallel solver exhibits fair scalability even when the data resides off-chip, a property that is not present for the simple CG targeted in our previous work.

The rest of the paper is structured as follows. In Sections 2 and 3 we briefly describe the task-parallel version of ILUPACK and the target architecture, respectively. In Section 4 we experimentally analyze the scalability of the solver and the impact of voltage-frequency scaling on the execution time. In Section 5, we repeat this analysis from the perspectives of power and energy consumption, so as to obtain an estimation of the energy gains (or losses) that would result from an execution that employed increasing levels of hardware concurrency. In Section 6 we link error corruption with degradation of convergence for the iterative solver, and we discuss its impact under two different scenarios. Finally, in Section 7 we offer a few concluding remarks.

2. Task-Parallel Implementation of ILUPACK

Consider the linear system $Ax = b$, where $A \in \mathbb{R}^{n \times n}$ is sparse, $b \in \mathbb{R}^n$, and $x \in \mathbb{R}^n$ is the sought-after solution. ILUPACK integrates an "inverse-based approach" to compute and apply an algebraic multilevel preconditioner in order to accelerate the iterative solution of the system [3]. In analogy with the HPCG benchmark, we consider hereafter that A is symmetric positive definite (s.p.d.), and we study the (preconditioned) iterative solution stage only, dismissing the computation of the preconditioner. The solve procedure involves a sparse matrix-vector product (SPMV), the application of the preconditioner, and a few vector operations (basically DOT products, AXPY-like updates and vector norms) per iteration [13]; see Figure 1. We emphasize that a similar PCG iteration underlies the HPCG benchmark.

Our task-parallel version of ILUPACK decomposes the solver into tasks, and abides to the existing inter-task dependencies in order to produce a "correct" (i.e., dependency-

[2] `http://ilupack.tu-bs.de`

$A \rightarrow M$	O0. Preconditioner computation
Initialize $x_0, r_0, z_0, d_0, \beta_0, \tau_0; k := 0$	
while $(\tau_k > \tau_{\max})$	Loop for iterative PCG solver
$w_k := Ad_k$	O1. SPMV
$\rho_k := \beta_k / d_k^T w_k$	O2. DOT product
$x_{k+1} := x_k + \rho_k d_k$	O3. AXPY
$r_{k+1} := r_k - \rho_k w_k$	O4. AXPY
$z_{k+1} := M^{-1} r_{k+1}$	O5. Apply preconditioner
$\beta_{k+1} := r_{k+1}^T z_{k+1}$	O6. DOT product
$d_{k+1} := z_{k+1} + (\beta_{k+1}/\beta_k) d_k$	O7. AXPY-like
$\tau_{k+1} := \| r_{k+1} \|_2$	O8. vector 2-norm
$k := k + 1$	
endwhile	

Figure 1. Algorithmic formulation of the preconditioned CG method. Here, $\tau_{\max}$ is an upper bound on the relative residual for the computed approximation to the solution.

aware) execution, while exploiting the task parallelism implicit to the operation. Concretely, each iteration of the PCG solve is decomposed into 8 macro-tasks, say O1–O8, related by a partial order which enforces an almost strict serial execution. Specifically, $O1 \rightarrow O2 \rightarrow O4 \rightarrow O5 \rightarrow O6 \rightarrow O7$, but O3 and O8 can be computed any time once O2 and O4 are respectively available. Here, each macro-task corresponds to one of the basic operations that compose the iteration: SPMV, application of preconditioner, DOT, AXPY and norms. The application of the preconditioner, performed inside O5, is decomposed into two groups of micro-tasks, both organized as binary trees, with bottom-up dependencies in the first one and top-down in the second, and a number of leaves l controlled by the user. The remaining macro-tasks are decomposed into l independent micro-tasks each, with the two dot products and vector 2-norm introducing a synchronization point each. For further details, see [2].

3. Setup

All the experiments in the paper were performed using IEEE double-precision arithmetic on an Exynos5 Odroid-XU development board assembled by Samsung. The Exynos5 contains an ARM Cortex-A15 quad-core cluster plus an ARM Cortex-A7 quad-core cluster integrated into a single big.LITTLE system-on-chip (SoC). Each Cortex-A7 core has a (32+32)–KByte L1 cache (data+instruction) and shares a 512–KByte L2 cache with its siblings. The system has 2 Gbytes of DRAM. In order to target low-power scenarios, we only employ the ARM Cortex-A7 cores during the experimentation, with frequencies that vary in the range of $\{250, \ldots, 600\}$ MHz with step changes of 50 MHz.[3] All codes were compiled using `gcc` version 4.8.1 with the appropriate optimization flags.

For the analysis, we employed a s.p.d. linear system arising from the finite difference discretization of a 3D Laplace problem, for a particular instance of size n=1,000,000 and nnz=6,940,000 nonzero entries in the coefficient matrix A. Thus, the data clearly resides

[3] The actual frequencies of the Cortex-A7 double those reported by the `cpufreq` driver and the real range is in $[500, 1200]$ MHz. The driver exposes these values (half of the actual) to make it easier to activate one of the clusters/deactivate the other by just re-setting the frequency. Otherwise, the ranges of Cortex-A15 and Cortex-A7 cores would overlap, and a different mechanism would be needed to change frequency across clusters.

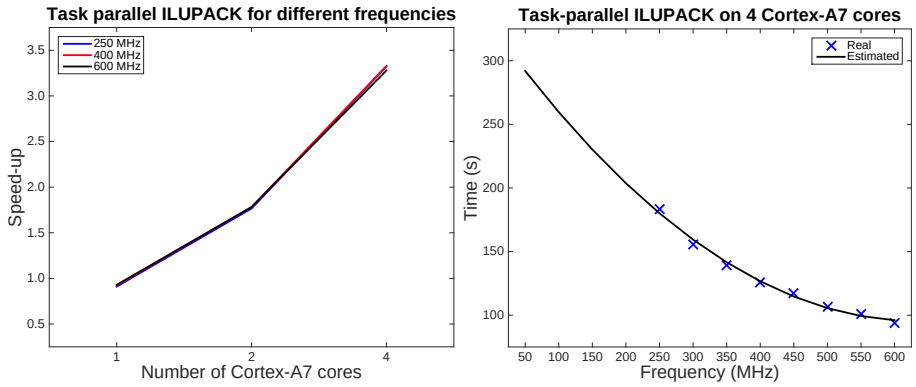

Figure 2. Computational performance using 1–4 (left) and 4 (right) ARM Cortex-A7 cores and a problem instance consisting of l=8 leaves.

off-chip. In the experiments, we always set $l = 8$, which offers enough parallelism for the four ARM Cortex-A7 cores available in the system.

4. Computational Performance vs Frequency-Voltage Scaling

In this section we analyze the scalability of our task-parallel solver based on ILUPACK, and the effect of frequency on performance.

The left hand-side plot in Figure 2 reports the speed-up of the solver with respect to the sequential version of ILUPACK, operating on the same problem instance, but with only one leaf. (Similar acceleration factors were observed when the sequential code was applied to the same problem with the data partitioned into 8 leaves, as done in the parallel case.) The results show fair speed-ups, close to $2\times$ for 2 Cortex-A7 cores and slightly below $3.5\times$ when the full quad-core Cortex-A7 cluster is employed. Furthermore, the acceleration is mostly independent of the operating (voltage-)frequency (pair). At this point, it is worth mentioning that our experiments with this code and a slightly larger instance of the same problem, on an Intel Xeon Phi, reveal accelerations factors of $15.5\times$ and $27.7\times$ with 16 and 32 x86 cores, respectively, providing additional evidence of the scalability of the solver [1].

The right-hand side plot in Figure 2 relates execution time vs frequency when the number of cores is fixed to 4; see also the columns labeled as "Time" in Table 1. In principle, we could expect that a variation of the operating frequency rendered an inversely proportional variation of the execution time. However, the interplay between frequency and time is more subtle and strongly depends on whether the code is compute- or memory-bound (i.e., dominated by the performance of the floating-point units or the access to memory, respectively) as well as the effect of frequency on the memory bandwidth. For example, in some x86 architectures, this bandwidth is independent of the frequency while in others, and specifically for the Cortex-A7 cluster, it is mildly governed by it. For example, in our experiments with the stream benchmark [12] on this architecture, we observed decreases in the memory bandwidth by factors 0.86 and 0.70 when the frequency was reduced in factors of 0.66 and 0.41.

Freq.	Time (in s)		Power (in W)				Energy (in J)			
(MHz)	$T(f_i)$		Total, $P_{\mathrm{T}}(f_i)$		A7, $P_{\mathrm{A7}}(f_i)$		Total, $P_{\mathrm{T}}(f_i)$		A7, $P_{\mathrm{A7}}(f_i)$	
f_i	Real	Estim.	Real	Estim.	Real	Estim.	Real	Estim.	Real	Estim.
50	–	291.8	–	0.215	–	0.118	–	62.5	–	30.9
100	–	259.5	–	0.214	–	0.111	–	55.8	–	27.2
150	–	230.1	–	0.217	–	0.109	–	50.3	–	24.4
200	–	203.6	–	0.225	–	0.112	–	46.1	–	22.6
250	183.4	180.0	0.237	0.239	0.120	0.121	43.5	43.0	21.9	21.7
300	155.8	159.3	0.263	0.259	0.141	0.137	40.9	41.2	21.0	21.7
350	139.6	141.5	0.287	0.287	0.161	0.161	40.1	40.6	22.5	22.6
400	125.7	126.6	0.324	0.325	0.193	0.194	40.7	41.2	24.3	24.5
450	117.4	114.6	0.368	0.372	0.235	0.238	43.2	43.1	27.5	27.3
500	106.7	105.5	0.436	0.431	0.297	0.292	46.6	46.1	31.7	31.1
550	100.6	99.3	0.502	0.502	0.359	0.359	50.5	50.4	36.1	35.7
600	94.0	96.0	0.587	0.587	0.438	0.438	55.2	55.8	41.2	41.3
Error	1.63e-2		6.29e-2		1.10e-2		9.20e-3		8.83e-3	

Table 1. Experimental results and estimations collected with the task-parallel version of ILUPACK using 4 ARM Cortex-A7 cores and a problem instance consisting of l=8 leaves. The last row ("Error") displays the average relative error $(\sum_i |r_i - e_i|/r_i)/n$, where r_i and e_i respectively denote the real and estimated values, $i \in \{50, 100, 150, \ldots, 600\}$, and n is the number of samples.

A major observation from this evaluation is that, for a complex code such as our task-parallel version of ILUPACK, there are fragments of the code that are compute-bound while others are memory-bound, making a prediction for the global behaviour of the application is difficult. Nevertheless, the plot shows that it is possible to apply linear regression in order to fit a quadratic polynomial that estimates the execution time T, as a function of the frequency f (in MHz): $T(f) = 5.80\text{E-}4\, f^2 - 7.33\text{E-}1\, f + 3.27\text{E+}2$ s. Moreover, the validity of this regression is quantitatively demonstrated by the coefficient of determination, $1 - r2 = 6.69\text{E-}3$, and the small relative differences between the real measures and the polynomial approximation shown in the row labeled as "Error" of Table 1. Using this polynomial to extrapolate the data, we observe an asymptotic lower bound on the execution time, independent of the operating frequency, close to 100 s.

5. Power and Energy Efficiency vs Frequency-Voltage Scaling

This section considers the following iso-power[4] scenario: Given the power dissipated by the system when operating at its highest frequency, what is the number of cores/clusters that can be crammed within the same power budget? In addition, given the variation of hardware concurrency that is possible under iso-power conditions, how this affects the execution time and, therefore, the energy efficiency of the alternative configurations?

The left-hand side plot in Figure 3 and (the columns labeled as "Power" in) Table 1 show the total power dissipated by the Exynos5 SoC plus the memory DIMMs as well as that of the Cortex-A7 cluster only when executing the task-parallel version of ILUPACK using 4 cores. For this metric, it is also possible to fit a precise linear regression curve.

[4] An analysis via a given *iso-metric* aims to obtain a metric-invariant set of analysis or design solutions.

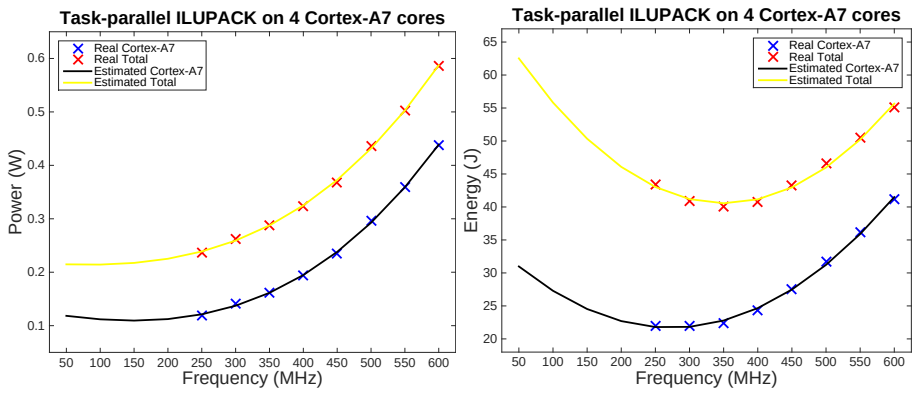

Figure 3. Power and energy efficiency vs frequency-voltage scaling using 4 ARM Cortex-A7 cores and a problem instance consisting of l=8 leaves.

For this purpose, we take into account that the power is cubically dependent on the frequency.[5] Thus, for example, the total power is accurately estimated by $P_T(f) = 1.46\text{E-}9\, f^3 + 2.76\text{E-}7\, f^2 - 7.60\text{E-}5\, f + 2.18\text{E-}1$ W, as $1 - r2 = 5.72\text{E-}4$. The regression curves expose asymptotic lower bounds on the power dissipation rates, at 0.21 W and 0.11 W respectively for the total and Cortex-A7 cluster, which can be attributed to the static power of this board/CPU.

To conclude this initial analysis, the right-hand side plot in Figure 3 and (the columns labeled as "Energy" in) Table 1 illustrate the behaviour of the code/architecture from the point of view of total and Cortex-A7 energy consumption. The former can be approximated by the quadratic polynomial $E_T(f) = 2.43\text{E-}4\, f^2 - 1.70\text{E-}1\, f + 7.04\text{E+}1$, with $1 - r2 = 7.36\text{E-}3$. (Alternatively, we could have used $E_T(f) = T(f)P_T(f)$.) This plot reveals a "sweet spot" (optimal) from the point of view of total energy at 350 MHz, which is shifted to 250 MHz if we consider the energy consumed by the Cortex-A7 cluster only.

For simplicity, let us work hereafter with the approximations to power and energy obtained via linear regression. Table 1 reports that the Exynos5 SoC plus the memory DIMMs, operating at the highest frequency ($f_{\text{M}} = 600$ MHz), dissipate $P_{\text{T}}(f_{\text{M}}) = 0.587$ W, with the Cortex-A7 cluster itself being responsible for a large fraction of this: $P_{\text{A7}}(f_{\text{M}}) = 0.438$ W. However, from the point of view of energy, the optimal configuration is attained when the application is executed with the cores at $f = 350$ MHz or 250–300 MHz, depending respectively on whether we consider the consumption by the complete SoC+DIMMs or the Cortex-A7 only.

Let us examine the intermediate case $f_{\text{opt}} = 300$ MHz. At this frequency, the Exynos5+DIMMs dissipate $P_{\text{T}}(f_{\text{opt}}) = 0.259$ W while the Cortex-A7 cluster is responsible for $P_{\text{A7}}(f_{\text{opt}}) = 0.137$ W. This implies that, with the same power (*iso-power*) dissipated by the system operating at f_{M}, we can feed an ideal configuration, which operates at f_{opt}, and consists of $P_{\text{T}}(f_{\text{M}})/P_{\text{T}}(f_{\text{opt}}) = 0.587/0.259 = 2.26\times$ or $P_{\text{A7}}(f_{\text{M}})/P_{\text{A7}}(f_{\text{opt}}) = 0.438/0.137 = 3.19\times$ more Cortex-A7 clusters/cores, depending

[5] We can identify two components in the power: static and dynamic. In practice, the static power depends on the square of the voltage V while the dynamic power depends on $V^2 f$, with the voltage itself depending linearly on the frequency [9].

respectively on whether we account for the full Exynos5+DIMMs or the Cortex-A7 cores only. Given that it is not possible to build a fragment of a cluster, we now approximate these numbers to $C_{\mathrm{T}}(f_{\mathrm{M}}, f_{\mathrm{opt}}) = \lfloor 2.26 \rfloor = 2$ and $C_{\mathrm{A7}}(f_{\mathrm{M}}, f_{\mathrm{opt}}) = \lfloor 3.19 \rfloor = 3$.

Let us analyze the energy efficiency of these options. At f_{M}, the system consumes $E_{\mathrm{T}}(f_{\mathrm{M}}) = 55.8$ J to solve the problem, with $E_{\mathrm{A7}}(f_{\mathrm{M}}) = 41.3$ J corresponding to the Cortex-A7 cluster. Assuming perfect scalability (see Section 4), we can expect that 2 Cortex-A7 clusters, operating at f_{opt}, obtain the solution in $\bar{T}(f_{\mathrm{opt}}, 2) = T(f_{\mathrm{opt}})/2 = 159.3/2 = 79.6$ s and 3 clusters in $\bar{T}(f_{\mathrm{opt}}, 3) = T(f_{\mathrm{opt}})/3 = 159.3/3 = 53.1$ s. For these configurations, we can therefore estimate total and Cortex-A7 energy consumption of $\bar{E}_{\mathrm{T}}(f_{\mathrm{opt}}, 2) = \bar{T}(f_{\mathrm{opt}}, 2) \cdot P_{\mathrm{T}}(f_{\mathrm{M}}) = 79.6 \cdot 0.587 = 46.7$ J and $\bar{E}_{\mathrm{A7}}(f_{\mathrm{opt}}, 3) = \bar{T}(f_{\mathrm{opt}}, 3) \cdot P_{\mathrm{A7}}(f_{\mathrm{M}}) = 53.1 \cdot 0.438 = 23.2$ J, respectively. This represents a gain in energy efficiency of $E_{\mathrm{T}}(f_{\mathrm{M}})/\bar{E}_{\mathrm{T}}(f_{\mathrm{opt}}, 2) = 55.8/46.7 = 1.19\times$ compared with the SoC+DIMMs; and $E_{\mathrm{A7}}(f_{\mathrm{M}})/\bar{E}_{\mathrm{A7}}(f_{\mathrm{opt}}, 3) = 41.3/23.2 = 1.77\times$ with respect to the Cortex-A7.

We recognize that our assumption of perfect scalability may be slightly too optimistic (tough we could expiremental observe a speed-up higher than 15 when executing the same code on 16 Intel Xeon cores). On the other hand, we point out that we considered that a system consisting of 2 (or 3) clusters dissipated the same instantaneous power as one composed of 2.26 (or 3.19) clusters. This partially compensates for the scalability approximation. Table 2 collects the results from this analysis for all possible frequencies, exposing the largest energy saving factors to be at $1.77\times$, for $f_i = 300$ MHz, if we consider only the Cortex-A7; and $1.34\times$, for $f_i = 350$ MHz, if we take into account the SoC+DIMMs.

Freq. (MHz) f_i	$P(f_{\mathrm{M}})/P(f_i)$		$C_i = C(f_{\mathrm{M}}, f_i) = \lfloor P(f_{\mathrm{M}})/P(f_i) \rfloor$		$\bar{T}(f_i, C_i) = T(f_i)/C(f_i)$		$\bar{E}(f_i, C_i) = \bar{T}(f_i, C_i) \cdot P(f_{\mathrm{M}})$		Gain/loss= $E(f_{\mathrm{M}})/\bar{E}(f_i, C_i)$	
	Total	A7	Total	A7	Total	A7	Total	A7	Total	A7
50	2.73	3.71	2	3	145.9	97.2	85.6	42.6	0.65	0.96
100	2.74	3.94	2	3	129.7	86.5	76.1	37.8	0.73	1.09
150	2.70	4.01	2	4	115.0	57.5	67.5	25.1	0.82	1.63
200	2.60	3.91	2	3	101.8	67.8	59.7	29.7	0.93	1.38
250	2.45	3.61	2	3	90.0	60.0	52.8	26.2	1.05	1.57
300	2.26	3.19	2	3	79.6	53.1	46.7	23.2	1.19	1.77
350	2.04	2.72	2	2	70.7	70.7	41.5	30.9	1.34	1.33
400	1.80	2.25	1	2	126.6	63.3	74.3	27.7	0.75	1.48
450	1.57	1.84	1	1	114.6	114.6	67.2	50.1	0.82	0.82
500	1.36	1.50	1	1	105.5	105.5	61.9	46.2	0.90	0.89
550	1.16	1.22	1	1	99.3	99.3	58.2	43.4	0.95	0.94
600	1.00	1.00	1	1	96.0	96.0	55.8	41.3	1.00	1.00

Table 2. Potential energy savings (or losses) within an iso-power budget scenario determined using 4 ARM Cortex-A7 cores and a problem instance consisting of l=8 leaves.

To summarize the analysis in this section, our experiments show that there exist some "sweet points" from the perspective of frequency, which could allow to leverage a larger number of cores that match the power budget of a full Cortex-A7 cluster operating at the highest frequency (iso-power), solve the problem in less time, and increase energy efficiency with respect to that configuration.

6. Energy Efficiency and Fault Tolerance

NTVC is a technology that promises important power/energy reductions by lowering the voltage while keeping the frequency mostly constant [11]. While this decrease of power can, in principle, be leveraged to integrate additional core concurrency, the potential negative effect is that these hardware becomes less reliable and additional mechanisms (e.g., in software) need to compensate for it.

In this section we study the error rate that can be accommodated into an iterative solver such as ILUPACK, running on a "faulty" architecture, while still being more efficient from the point of view of energy consumption than the same code executed on a reliable system. For this purpose, we consider that an unreliable hardware, operating under NTVC, only corrupts the results from the floating-point arithmetic operations, degrading the convergence rate of the PCG iteration. In practice, faults can occur anywhere, producing an ample variety of effects: from catastrophic crashes in the program (which are easy to detect) to soft errors exerting no visible impact on the application (e.g., a fault that corrupts the result of a branch prediction may affect performance, but not the numerical results) [15].

As argued earlier, we can decompose the power into its static and dynamic components, where the global consumption depends on $V^2 f$ and, for safety, current technology sets V proportionally to f. Assume that NTVC can lower the voltage while maintaining the frequency. (In practice, even with NTVC, the frequency would need to be slightly reduced as the voltage is diminished.) In an ideal situation where no errors occur, we can therefore expect that the execution time of the solver does not vary, but its power dissipation rate rapidly decays (indeed, quadratically with the reduction of voltage) and, therefore, so does energy (in the same proportion). In other words, scaling the voltage as V/σ changes the power to P/σ^2, but preserves the execution time T, hence reducing the energy by E/σ^2. Let us analyze next the trade-offs in two configurations where errors are present:

- **Improve power/energy at the expense of time-to-solution (TTS).** Given a decrease of power $\hat{P} = P/s$ (produced by a reduction of voltage by $\sigma = \sqrt{s}$), we can afford an error rate $\hat{e}$ which degrades convergence by a factor of up to $\hat{d}$, with $\hat{d} \leq s$, with an analogous increase in execution time $\hat{T} = T \cdot \hat{d}$, and still save energy by a factor $\hat{d}/s$, as $\hat{E} = \hat{T} \cdot \hat{P} = T \cdot \hat{d} \cdot P/s = (\hat{d}/s)E \leq E$.

- **Improve TTS/energy using additional hardware.** Alternatively, we can leverage a reduction of power in P/s to increase the number of computational resources (i.e., hardware) by a factor of s, and thus confront an iso-power scenario with a dissipation rate $\tilde{P} \approx P$. This approach would benefit from the near perfect scalability of ILUPACK, in order to reduce the execution time to T/s (when no errors are present). In addition, we require that the increase in the error rate induced by the use of additional hardware, likely from e to $\tilde{e} = e \cdot s$, does not produce an analogous raise the degradation rate by a factor of s as, otherwise, the advantages of increasing the amount of hardware will be blurred. Our only hope in this sense is to introduce a low-cost mechanism in the solver, in the form of a detection+correction or a prevention strategy [10,14] which, for a cost $\tilde{c}$ that depends on the iteration number, limits the effect of the errors on the convergence degradation.

Under these ideal conditions, with a convergence degradation factor of $\tilde{d} \leq s$, the execution is completed in $\tilde{T} = (T/s + \tilde{c}) \cdot \tilde{d} = (\tilde{d}/s)T + \tilde{c}\tilde{d} \leq T$; and the energy consumed by the new configuration is given by $\tilde{E} = \tilde{T} \cdot \tilde{P} = (T/s + \tilde{c}) \cdot \tilde{d} \cdot P = (\tilde{d}/s)E + \tilde{c}\tilde{d}P \leq E$. The factor $\tilde{c}\tilde{d}$ in the time and energy expressions represents the cost of the fault tolerance mechanism, which initially increases time by $\tilde{c}$ units, but in total raises the cost to $\tilde{c}\tilde{d}$ because of the convergence degradation. For simplicity, for the energy expression we assume that this mechanism dissipates the same power rate as the solver, though the formula can be easily adapted otherwise.

7. Concluding Remarks

We have conducted an experimental analysis of the effect of voltage-frequency scaling on the scalability, power dissipation and energy consumption of an efficient multi-threaded version of the preconditioned CG method on a low-power ARM multicore SoC. Our results show a remarkable scalability for the solver, independent of the operating frequency f, but a lower bound on the execution time as f grows. Combined with the cubic relation between power and frequency, this determines an optimal operating frequency from the point of view of energy consumption which is in the low band, much below the highest possible in this chip. Using linear regression to interpolate the experimental data, we have exposed the potential gains that can be obtained by leveraging the power budget available by running at a lower frequency to execute the code with an increased number of cores. For example, under certain conditions, the execution of the solver using 2 ARM Cortex-A7 clusters at a very low frequency can report energy savings around 16% with respect to an execution on a single cluster running at the highest frequency while matching the iso-power of the global SoC+DIMMs. If we consider the power budget for the Cortex-A7 cluster, though, it is possible to employ 3 ARM Cortex-A7 clusters instead of 1, and the savings boost to 43%.

NTVC promises important reductions in the power consumption that allow the exploitation of Moore's law to build multicore processors with increased numbers of cores (or wider SIMD units) at the expense, possibly, of higher error rates. For an iterative numerical solver like ILUPACK, we can expect that many of these errors occur during the floating-point arithmetic, affecting the convergence rate of the method. In this sense, it is important that the convergence degradation of the method does not grow at the same pace as the error rate/number of computational resources enabled by NTVC. Otherwise, benefits are cancelled, and we can only trade lower energy consumption for increased execution time while keeping constant the number of computational resources.

Acknowledgments

The researchers from *Universidad Jaume I* (UJI) were supported by projects TIN2011-23283 and TIN2014-53495-R of the Spanish *Ministerio de Economía y Competitividad*. We thank Maria Barreda from UJI for her support with the power measurement system on the Exynos5 SoC. This research has also been partially supported by the UK Engineering and Physical Sciences Research Council (grants EP/L000055/1,

EP/L004232/1, EP/M01147X/1, EP/M015742/1, EP/K017594/1), the Royal Society (grant WM150009), and the European Commission (grants 644312, 610509, 323872).

References

[1] J. I. Aliaga, R. M. Badia, M. Barreda, M. Bollhöfer, and E. S. Quintana-Ortí. Leveraging task-parallelism with OmpSs in ILUPACK's preconditioned CG method. In *26th Int. Symp. on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 262–269, 2014.

[2] J. I. Aliaga, M. Bollhöfer, A. F. Martín, and E. S. Quintana-Ortí. Exploiting thread-level parallelism in the iterative solution of sparse linear systems. *Parallel Computing*, 37(3):183–202, 2011.

[3] Matthias Bollhöfer and Yousef Saad. Multilevel preconditioners constructed from inverse-based ILUs. *SIAM Journal on Scientific Computing*, 27(5):1627–1650, 2006.

[4] C. Chalios, D. S. Nikolopoulos, S. Catalán, and E. S. Quintana-Ortí. Evaluating asymmetric multicore systems-on-chip and the cost of fault tolerance using iso-metrics. *IET Computers & Digital Techniques*, 2015. To appear.

[5] R.H. Dennard, F.H. Gaensslen, V.L. Rideout, E. Bassous, and A.R. LeBlanc. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE J. Solid-State Circuits*, 9(5):256–268, 1974.

[6] J. Dongarra and M. A. Heroux. Toward a new metric for ranking high performance computing systems. Sandia Report SAND2013-4744, Sandia National Laboratories, June 2013.

[7] M. Duranton, K. De Bosschere, A. Cohen, J. Maebe, and H. Munk. HiPEAC vision 2015, 2015. `https://www.hipeac.org/assets/public/publications/vision/hipeac-vision-2015_Dq0boL8.pdf`.

[8] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Proc. 38th Annual Int. Symp. Computer Arch.*, ISCA'11, pages 365–376, 2011.

[9] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Pub., 5th edition, 2012.

[10] M. Hoemmen and M. A. Heroux. Fault-tolerant iterative methods via selective reliability. In *Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2011.

[11] U.R. Karpuzcu, Nam Sung Kim, and J. Torrellas. Coping with parametric variation at near-threshold voltages. *Micro, IEEE*, 33(4):6–14, 2013.

[12] J. D. McCalpin. STREAM: sustainable memory bandwidth in high performance computers.

[13] Y. Saad. *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 3rd edition, 2003.

[14] P. Sao and R. Vuduc. Self-stabilizing iterative solvers. In *Workshop Latest Advances in Scalable Algorithms for Large-Scale Systems*, pages 4:1–4:8, 2013.

[15] Daniel J. Sorin. *Fault Tolerant Computer Architecture*. Morgan & Claypool Pub., 2009.